DOCUMENT RESUME

ED 068 068 HE 003 521

AUTHOR	Etnyre, Vance; And Others
TITLE	GENIRAS. Level-1 User's Manual.
INSTITUTION	Illinois Univ., Urbana. Coll. of Engineering.
PUB DATE	May 72
NOTE	90p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computer Programs; *Computers; *Educational Administration; Guides; *Higher Education; *Management Information Systems; Manuals; *Programing Languages

ABSTRACT

GENIRAS (Generalized Information Storage, Retrieval, and Application System) is the name given to a computer-aided information system designed for the storage, retrieval, analysis, manipulation, and display of records, files, and associated data. It is intended to characterize the integration of the data storage and retrieval functions of generalized file management systems with the application functions advocated for management information systems. This manual is concerned with informing the user of how to operate a keyboard terminal; how to use the PLORTS (PL One Remote Terminal Subset) system to create, update, combine, and display input files of instructions; and how to construct correct statements in GENERAL, the input language of GENIRAS. (Author/HS)

ED 068063

GENIRAS

LEVEL-1 USER'S MANUAL

Prepared by
Vance Etnyre
Charles Ludmer
Kit McMillion

U.S. DEPARTMENT OF HEALTH
EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

HEW 3524

May 1972

FILMED FROM BEST AVAILABLE COPY

ED 068068

GENIRAS

LEVEL-1 USER'S MANUAL

Prepared by
Vance Etnyre
Charles Ludmer
Kit McMillion

May 1972

ACKNOWLEDGMENTS

The GENIRAS system originated in the College of Engineering at the University of Illinois at Urbana-Champaign with the purpose of aiding administrators in various forms of institutional planning. Participating with the investigator, V. A. Etnyre, in the original formulation of the system were A. F. Graziano, Assistant Vice-Chancellor for Academic Affairs; J. J. Desmond, Associate Director, Engineering Experiment Station; and M. Davenport, currently Vice-President, Texas Tech University.

Arrangements for the original funding of the project were made with the help of H. E. Carter, Vice-Chancellor for Academic Affairs at the time; A. F. Graziano; and R. J. Martin, Director, Engineering Experiment Station.

As the project progressed, design and programming were coordinated with the help of J. H. Stanley, Research Programmer and Director, Commerce Computing Center.

Among those contributing to the applications programs were W. Montgomery, M. Walker, E. Domke, M. O'Connor, R. Palmer, and G. Foat. Systems programs for the later versions of GENIRAS were written by J. H. Stanley.

Funding was coordinated by A. F. Graziano, H. E. Carter, D. C. Drucker, Dean of Engineering, R. J. Martin, J. J. Desmond, and H. L. Wakeland, Associate Dean of Engineering.

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
II.	DESCRIPTION OF FILES	5
III.	GENIRAS/PLORTS INSTRUCTIONS	7
IV.	GENIRAL LANGUAGE STATEMENTS	10
	Data Identifiers	13
	Connectors	16
	Verbs	20

APPENDICES

I. INTRODUCTION

GENIRAS is the name given to a computer-aided information system designed for the storage, retrieval, analysis, manipulation, and display of records, files, and associated data. The acronym GENIRAS, pronounced like 'generous' is derived from the words GENeralized Information Storage, Retrieval, and Application System. It is intended to characterize the integration of the data storage and retrieval functions of generalized file management systems with the application functions advocated for 'management information systems.'

GENIRAS is implemented on the ILLINET computer network at the University of Illinois, using the remote file handling and job entry facility of the PLORTS system. PLORTS, which stands for PL One Remote Terminal Subset, enables the user to access the ILLINET computer system, through a keyboard terminal like a teletypewriter. The physical components of the PLORTS system include the terminal and coupled communication lines, a communications multiplexor computer (PDP-7), a filing, editing, and remote job entry facility (currently implemented on an IBM 360/75), and a batch processing computer system (the IBM 360/75). Two types of terminals--the IBM 2741 terminal and the Teletype Model 33 terminal--through which the user communicates with the other parts of the system are described in Appendix B of this document.

The following assumptions about the nature of data

and data handling lead to the definition and development of the GENIRAS system.

1. Most data are recorded in meaningful groups or structures rather than in random pieces. A data group is a collection of data items which share one or more common attributes. An attribute might describe an organizational boundary, a specific time period, or a specific functional relationship. For instance, a particular data group might share the common attribute that they represent costs, so the name COSTS might be used as a reference name within the system. The group of costs, in turn, might be indexed according to the years during which the costs occurred and according to the departments which incurred them. Each group of cost data sharing a common index (year or department) would be a subgroup of the larger group.

2. Most data processing requires that a particular operation or procedure be performed with each member of a data group. For example, a particular group of costs might be summed, averaged, or compared. The process of assuring that every member of a data group is included in the procedure is called exhausting the data group.

3. Data processing frequently requires the use of corresponding elements from different data groups. The corresponding elements are those elements that

share one or more specified attributes. For example, the information contained in several different data groups, but applying to a particular year, might be desired. The process of finding the corresponding elements is called matching.

4. Many individuals understand their data processing needs well enough to describe them easily and precisely, in conversational English, to a friend or colleague. However, these individuals are unable to use the power of a computer to help solve their problems because typical computer languages are so completely different from conversational English.

5. Every specific capability that may be desired in a data processing system cannot be anticipated. New processing mechanisms may become necessary, and those already in existence may have to be applied to newly created data groups.

GENIRAS systems allow the user to interact at various levels with a computer-based information system. At the lowest level of sophistication (Level 1), the standard procedures supplied by GENIRAS can be applied directly to a user's established collection of data. Existing procedures are specified with predefined, "English-like" instructions to satisfy simple data retrieval and processing needs. At this level, matching of data items and exhausting of data groups are handled automatically by the system. At a more sophisticated level (Level 2), new sets of procedures may

be defined using the English-like language of the system, the full power of a procedural language (PI/I, COBOL, or FORTRAN), basic assembly language, or combinations of these languages. Special purpose languages also may be defined by a user for any particular application.

GENIRAS takes advantage of the structure of data groups to relieve the user of the need to match corresponding items or to exhaust data groups. These functions are handled automatically. Procedures for copying, sorting, updating, and transforming data also are supplied as a convenience to the user. GENIRAS permits a great deal of ease, power, and flexibility in defining and using collections of data, in applying procedures to collections of data, and in employing languages to define these procedures.

This manual will be concerned with the Level 1 use of the system. At this level, the user must understand how to operate a keyboard terminal, how to use the PLORTS system to create, update, combine, and display input files of instructions, and how to construct correct statements in GENIRAL, the input language of GENIRAS.

II. DESCRIPTION OF FILES

Using GENIRAS implies nothing more than manipulating two different kinds of files. There are data files consisting of collections of records stored on peripheral memory devices of the computer, and there are PLORTS files of commands and descriptive material. The distinction between these two kinds of files is of basic importance because they are created and manipulated in completely different ways.

A data file consists of a series of records designed for processing by a computer. Each record is composed of fields containing individual pieces of information, and these fields can be identified by data names. To retrieve information using GENIRAS, the computer needs only the identifying name of a data field. Use of the system at Level 1 assumes the existence of all necessary data files.

The other kind of file--the PLORTS file--may hold data, job control information, descriptive material, and instructions to the computer for the manipulation of data files. When a job is to be executed, a PLORTS file containing the job instructions is sent to the computer. Each PLORTS file is given a name of eight or less characters, but not including blanks, commas, or periods. PLORTS files are opened from a user's terminal by typing the command OPEN followed by a single space and the name of the file. After a PLORTS file has been opened, its contents may be altered or listed at the terminal. When a PLORTS file is

closed, the instructions it contains can be passed to the computer for execution by giving the RUN command at the terminal. The section of this manual dealing with the operation of the terminals gives a more complete description of the procedures for manipulating PLORTS files.

The following sections will deal with the creation, updating, and display of PLORTS files. The use of the GENIRAL language to create programs for storage in PLORTS files and the instructions necessary to cause execution of these programs also will be discussed.

III. GENIRAS/PLORTS INSTRUCTIONS

The instructions sent to the computer system from a terminal fall into three basic categories on the basis of the portion of the system which they affect. The first level of instructions is intended for the creation, maintenance, and editing of PLORTS files. Some of these instructions cause operations to be performed on entire PLORTS files, and are used when the terminal is in the closed condition. Other instructions are used to create or alter the contents of a particular file, and are used when the terminal is in the open condition. (If any PLORTS file is open for examination or alteration, the terminal is said to be open.) Also included in the first category are multiplexor functions which assist in the formatting and entry of job information. These commands are described in the section of this manual dealing with the use of the terminals.

If the terminal is closed, as it is immediately after signing on, the commands available can cause the names of stored files to be listed, can cause an entire PLORTS file to be deleted, can open a specific file, or can cause the program in a particular file to be run.

When a specific file is open (only one file can be open at any time), its entire contents may be listed, individual lines may be listed, and lines may be entered or deleted. As an example, if the command OPEN BASICS is typed at the terminal, the PLORTS file called BASICS will be opened. If no such file exists, a new file by that name

will be created. The LIST command can be used to display the entire contents of the file. If the contents of BASICS were printed out, each line of the text would constitute a record stored in the file by its line number.

The records in a file are stored by line number in increasing numerical order. Line numbering can be accomplished manually or automatically through one of the built-in multiplexor functions. If a line with a particular number is entered into an open file, it will be inserted automatically in its proper numerical position. If a line with that number already exists, it will be replaced by the new line. Commands for both the open and closed terminal conditions are explained in detail in the section of this manual dealing with the operation of the terminals.

The second level of instructions to GENIRAS are those written in GENIRAL (GENeral Information Retrieval and Application Language), the system input language. Statements in GENIRAL direct the computer system to display the contents of various data files, or to manipulate stored data for the derivation of new, useful information. The parts of a GENIRAL statement are the verb, the data names, modifiers of the data names to be used, and connectors. Because GENIRAL statements are the means for data processing with GENIRAS, their composition and their use are discussed in much greater detail in the following section.

The final level of instructions, in Job Control Language (JCL), gives information about the particular memory

devices where data is stored. Since JCL is involved primarily with what might be called the administrative details of jobs, it is secondary to the purpose of GENIRAS. Frequently employed JCL statements are available as prewritten packages for insertion in PLORTS files. Consequently, a detailed discussion of JCL has not been included in this manual.

IV. GENIRAL LANGUAGE STATEMENTS

Input statements for GENIRAS are written with the simple English-like sentence structure that is basic to the language, GENIRAL. They are designed to be straightforward and easy to learn. Each statement contains a main clause, composed of a verb and one or more data identifiers, and ends with a period. Data names or identifiers--the nouns of GENIRAL--are separated by words and symbols called connectors and may be modified with qualifiers.

For example, the GENIRAL statement

```
ADD SALARIES_AND WAGES TO EQUIPMENT_IMPROVEMENT
GIVING TOTAL_EXPENSES FOR DEPARTMENT (2255).
```

uses SALARIES_AND_WAGES,EQUIPMENT_IMPROVEMENT, and TOTAL_EXPENSES as data indentifiers. It is obvious from the statement that the first two data identifiers refer to existing data while the third identifier is the name to be given to the result of the addition. The single verb ADD begins the statement. The connector FOR near the end of the statement indicates that all the existing data to be located and all the new data to be filed must be indexed with the department code number 2255.

The verb of an input sentence is chosen from a list of verbs that have been defined for the particular GENIRAS installation. The verb indicates what the computer is to do with the specified data groups. Each verb and an explanation of its correct use is described in a later section. Verbs for GENIRAL can be added to or deleted from the system at any time. Thus, each institution using a GENIRAS

system can define its own verbs as necessary.

Data identifiers also may be defined by the individual user, or by the institution maintaining the system. Data identifiers are the names a user employs when he wants to refer to the information stored in particular fields of some data file.

In some statements the main clause may be followed by a single options clause containing any specified options for the verb in the statement, or containing modifiers applying to all the data identifiers in the statement. If an options clause is included in a statement, it is separated from the main clause by a semicolon. At the end of certain types of statements, after the period, a data section also may be appended to enter data into the system. A table illustrating the syntax of GENIRAL is included as Appendix A to this manual. The table illustrates, in a formal way, the possible arrangements, including punctuation, of the 'parts of speech' of GENIRAL.

The following examples show statements consisting only of main clauses.

```
DEFINE DATA FROM NAMES:COSTS.

ADD SALARY_COSTS, EQUIPMENT_COSTS, RESEARCH_COSTS,
   AND MAINTENANCE_COSTS GIVING TOTAL_COSTS.

PERFORM COST_ANALYSIS.

LIST DEPARTMENT, YEAR, AND TOTAL_COSTS.
```

The first example identifies the file containing all the data names mentioned in the other main clauses. The

EXECUTE

The verb EXECUTE causes the execution of a specific prewritten Level-2 procedure stored in GENIRAS. The options applying to EXECUTE depend entirely on the options defined for the procedure being called. The creation of Level-2 procedures and their options is discussed in the supplement to this manual dealing with Level-2 use of the system.

The verb EXECUTE is employed through a GENIRAL statement of the form:

EXECUTE ----- USING -----, -----,... AND ----- GIVING -----.

The first blank in the statement contains the name of the prewritten procedure. If the procedure has been defined to use arbitrary data groups, the data names for a specific application must be included in the EXECUTE statement with the connector USING. If a new data group is created by the procedure being called, the name for the new group must follow the connector GIVING in the statement.

PRINT, another GENIRAL verb, can be used in place of EXECUTE if the procedure being called is designed to create a printed report. On the other hand, EXECUTE can be substituted for PRINT, wherever it appears.

The following example illustrates the use of the verb EXECUTE:

```
EXECUTE ATTRITION_ANALYSIS FOR CURRIC (2250) USING
   NEW_FRESHMEN.
```

This statement assumes the existence of a prewritten, Level-2 procedure called ATTRITION_ANALYSIS. The analysis apparently

has been written to operate on whichever data group is specified in the statement after USING. In this case, the data group is NEW_FRESHMEN enrollment in curriculum 2250, where curriculum and year have been defined as the index attributes of enrollment data.

second example causes the addition of automatically matched data items. The third example causes the performance of a prewritten procedure called COST_ANALYSIS (see verb DEFINE for procedures). Finally, the fourth example gives a listing, in three columns, of the department and the year to which particular cost totals apply and of the cost totals, themselves. The use, in this example, of the names DEPARTMENT and YEAR assumes the prior identification of the index attributes with these names.

The examples which follow illustrate the placement of modifiers with connectors in the main clause and as option assignments in the options clause.

```
ADD SALARY_COSTS, EQUIPMENT_COSTS, RESEARCH_COSTS,
   AND MAINTENANCE_COSTS GIVING TOTAL_COSTS FOR
   (2250,1970).

ADD SALARY_COSTS, EQUIPMENT_COSTS, RESEARCH_COSTS,
   AND MAINTENANCE_COSTS GIVING TOTAL COSTS;
   FIRST_INDEX=2250, SECOND_INDEX=1970.

ADD SALARY_COSTS (2250,1970), EQUIPMENT_COSTS
   (2250,1970), RESEARCH_COSTS (2250,1970), AND
   MAINTENANCE_COSTS (2250,1970) GIVING TOTAL_COSTS
   (2250,1970).
```

All three of the examples are equivalent. The connector FOR is used in the first example to apply the desired index attributes for all the data items. In the last example, the index attributes are applied individually to each data item. If one of the index attributes is not going to be specified, one way to indicate the missing attribute is to place a single asterisk where the attribute would have

been within the parentheses of the first or the third example.

In the options clause, the specification of options that apply to the verb is accomplished in the same way as the specification of options that are modifiers. The options available with each of the verbs defined for GENIRAS are described later in this manual with the verbs to which they apply.

Only very few of the verbs defined for GENIRAS permit the use of a data section. When a data section is permitted, the nature of its contents is included with the description of the verb to which it applies. The form for the data section is as follows:

```
*BEGIN (optional name)
the statements of the data section
*END
```

Data Identifiers

Data names or identifiers can be any combination of letters, special symbols, and numbers. If blanks or special symbols are used within a data name, the entire name must be enclosed by single quotation marks each time it is referenced. Although PLORTS file names are limited to eight or less characters, and are not to include blanks, commas, or periods, this is not the case with data identifiers. Data names may be of any length, as long as the first twenty characters are unique to that name.

Some examples of acceptable names are:

```
OVERTIME_WAGES
'OVERTIME WAGES'
```

```
'OVERTIME WAGES FOR NON-PERMANENT WORKERS'
OVRTWG
'OW*.@¢ZI#L'
```

The difference between the first example and the second example is of particular importance. In the first example, the words are connected by an underscore. In the second example, the identifier must be enclosed with quotation marks because the words are separated by a space. These two forms of the same name are not equivalent. Once the form of a data name has been chosen, that form must be used wherever the name appears in subsequent programming. However, two or more names may be assigned to the same data item, allowing the use of full descriptive names when clarity is important, and the use of shorter names for speed and convenience. Data names may be added or changed at any time.

Modifiers in GENIRAL can be used in several ways to qualify the description of data items. Modifiers include file names and subfile names (to distinguish between two items of the same name coming from different files), index attributes, and selection criteria. Index attributes are modifiers used to specify a particular subset of the data identified by a data name and a particular index value. Selection criteria are more complex modifiers that can include many specifications consisting of names, values, and logical relationships.

If two data items have the same name, but come from different files, the addition of the appropriate file name

is sufficient to uniquely identify each item. When a name is used within several subfiles, both the file and subfile names must be added to the data name. Examples of correct specifications follow:

```
DATANAME:FILE1
DATANAME:FILE2
DATANAME:FILE1:SUBFILE1
DATANAME:FILE1:SUBFILE2
```

By providing index attributes and selection criteria, GENIRAS allows users to reference either entire data groups or subgroups of data items. An entire data group consists of all items having the same name. A subgroup consists of only those items in the specified group which have a certain common attribute or identifying characteristic. For example, if costs were indexed by the year of their occurrence, then COSTS (1969) and COSTS (1970) would designate two subgroups within the group named COSTS.

No more than two index attributes may be defined for each file, subfile, or data item. The number of definable attributes may be increased in the future. The description of data and the application of attributes is described more completely under the description of the verb DEFINE.

Selection criteria are employed similarly in the discrimination and selection of data items. They also describe subgroups of the data associated with a data name, but they can be defined much more flexibly. For instance, the data referred to by the identifier COSTS (1970,DEPT#2, CRITERION=EQUIPMENT_COST) is limited, in order, by the two

index attributes, 1970 and DEPT#2, and by the criterion EQUIPMENT_COSTS for a very restricted subgroup of the group of costs.

Connectors

The connectors used in GENIRAL serve two purposes. They help to make the input statements more readable, and they provide information about the use of data items during the execution of an input command. Certain connectors specify that a particular data item mentioned in a statement is to control a process such as matching. Other connectors specify that a new data item will be created by the execution of an input command.

To make the discussion of connectors more meaningful, it is helpful to introduce the concept of the action control block (ACB). An ACB is formed for each input statement in GENIRAL, and gives the GENIRAS system a uniform method for interpreting the instructions contained in each statement. The connectors in GENIRAL tell the system how a statement is to be translated into an ACB. The GENIRAS system then uses the contents of the ACB to control whatever process is requested.

An empty ACB has the following form:

Verb = -----

	Data name	First index	Second index	Criterion	File name	Subfile name
0	-----	-----	-----	-----	-----	-----
1	-----	-----	-----	-----	-----	-----
2	-----	-----	-----	-----	-----	-----
3	-----	-----	-----	-----	-----	-----
4	-----	-----	-----	-----	-----	-----
5	-----	-----	-----	-----	-----	-----
6	-----	-----	-----	-----	-----	-----
7	-----	-----	-----	-----	-----	-----
8	-----	-----	-----	-----	-----	-----
New	-----	-----	-----	-----	-----	-----

The number of data items (the number of horizontal rows) in an ACB will become as large as necessary for any application.

A connector from one of the lists shown below causes a data name and its modifiers to be placed in a particular location in the ACB. From a statement, the first data identifier, unpreceded by a connector, is placed in location 1 of the ACB.

The connectors	AND ,(comma) TO WITH BY	cause the next data identifier to be placed in the next available location in the ACB.
The connectors	FROM USING AGAINST MATCHING VERSUS	indicate that the next data item describes a control item. The next data name will be placed in location 0 in the ACB and subsequently will be interpreted as the control item by the system.
The connectors	INTO ONTO GIVING ON	indicate that a new file or data item is to be created under the next data name. The next name will be placed in the last location in the ACB and subsequently will be assigned to the new data created when the command is executed.
The connector	FOR	followed by one or more modifiers, will affect all the data items mentioned in a statement.

For example, if ITEM1 AND ITEM2 are the names of data items stored, respectively, in SUBFILE1 and SUBFILE2, and if the command

ADD ITEM1:FILE1:SUBFILE; TO ITEM2:FILE1:SUBFILE2
GIVING ITEM3 FOR (2250,1970); CRITERION='PDQ'.

is given, the following ACB will be created.

Verb = ADD

	Data name	First index	Second index	Criterion	File name	Subfile name
0	-----	-----	-----	-----	-----	-----
1	ITEM1	2250	1970	PDQ	FILE1	SUBFILE1
2	ITEM2	2250	1970	PDQ	FILE1	SUBFILE2
3	-----	-----	-----	-----	-----	-----
4	-----	-----	-----	-----	-----	-----
5	-----	-----	-----	-----	-----	-----
6	-----	-----	-----	-----	-----	-----
7	-----	-----	-----	-----	-----	-----
8	-----	-----	-----	-----	-----	-----
New	ITEM3	2250	1970	PDQ	-----	-----

In response to this ACB, GENIRAS will perform the following activities. First, the existing data items will be located. Only data items having the specified attributes (2250,1970) and satisfying criterion PDQ will be used in making calculations. Second, the data items named ITEM1 and ITEM2 will be added term by term to create a new data item called ITEM3. Because the index attributes and the criterion affect every data item named in the statement, the new data created under ITEM3 also will be indexed according to those attributes and that criterion.

As a second example, the statement GRAPH SALARIES AND EQUIPMENT_EXPENSES FOR DEPT (2220) AGAINST YEAR. will cause the following ACB to be created.

Verb = GRAPH

	Data name	First index	Second index	Crite-rion	File name	Subfile name
0	YEAR	2220	-----	-----	-----	-----
1	SALARIES	2220	-----	-----	-----	-----
2	EQUIPMENT_EXPENSES	2220	-----	-----	-----	-----
3	-----	-----	-----	-----	-----	-----
4	-----	-----	-----	-----	-----	-----
5	-----	-----	-----	-----	-----	-----
6	-----	-----	-----	-----	-----	-----
7	-----	-----	-----	-----	-----	-----
8	-----	-----	-----	-----	-----	-----
New	-----	-----	-----	-----	-----	-----

In response to the command in this example, a graph will be created with the values of the control item YEAR, along the horizontal axis, plotted against the corresponding values of the SALARIES and EQUIPMENT_EXPENSES for each year. The fact that YEAR is the control item is indicated in the statement by the connector AGAINST. The inclusion in the statement of DEPT, presumably defined as the name of the first index attribute, is optional. DEPT has been included here to increase the clarity of the statement since only one index attribute has been used.

It is not necessary for a level 1 user to memorize the structure of the action control block. It is only necessary to know: that data identifiers following the connectors FROM, USING, AGAINST, MATCHING, or VERSUS will be interpreted as control names; that identifiers following the connectors INTO, ONTO, GIVING, or ON will be assigned to newly created items; and that qualifiers following the connector FOR will be applied to all the data items mentioned in the input statement.

Verbs

The verbs presently available in GENIRAL will be described in terms of their use with an existing collection of data files and subfiles. Use of GENIRAS at Level 1 presupposed the existence of such an organized set of files.

In many cases, the index attributes of numerical department or curriculum code and year have been defined for the filed data. Other index attributes and criteria defined for the data will be explained as they appear in the examples.

The following example shows the use of the verbs ADD, LIST, and DEFINE.

```
 1 /*ID PS=3743,DEPT=ENGADM,NAME=GENIRAS
 2 //*       This job creates a data group called
 3 //*  DEPARTMENT_EXPENSES by adding together items from
 4 //*  the groups called SALARIES_AND_WAGES,
 5 //*  EQUIPMENT_IMPROVEMENT, and OTHER_EXPENSES.  After
 6 //*  the addition is performed, the department expenses
 7 //*  are listed.  The code (2250) is for the Department
 8 //*  of Metallurgy and Mining Engineering.
 9 // EXEC GENIRAS
10   DEFINE DATA FROM NAMES:EXPENSES.
11   ADD SALARIES_AND_WAGES, EQUIPMENT_IMPROVEMENT,
12      AND OTHER_EXPENSES FOR DEPT (2250) GIVING
13      DEPARTMENT_EXPENSES.
14   LIST DEPARTMENT_EXPENSES, SALARIES_AND_WAGES,
15      EQUIPMENT_IMPROVEMENT, AND OTHER_EXPENSES FOR
16      DEPT (2250).
17 //ENGDATA DD DSNAME=USER.P1191.ENGDATA,DISP=OLD
```

If the statements shown above are placed in a PLORTS file and the PLORTS file is run, the operations described in the program will be performed by the computer.

The first line of the program is an identification (ID) statement.* Although it and several other prewritten ID statements which are suitable to GENIRAS are stored in a PLORTS file called ID, instructions for preparing special ID statements are given in Appendix C.

The verb DEFINE is employed in this short program (line 9) to tell the computer that the data names used in the rest of the program refer to items from the data file called EXPENSES. Through the next statement in the program, the verb ADD creates a new data group from the existing data groups. However, the new data group is not stored permanently in the computer memory; it is destroyed when the job is completed unless special methods (see the verb SAVE) are employed to preserve it.

As the last specified operation in the program, the verb LIST provides a printout of the data groups mentioned in the LIST command (lines 12 and 13). Since the LIST and ADD commands in this example have been modified by the index attribute (2250), only the data applying to the Department of Metallurgy and Mining Engineering will be added and printed.

---

*No specifications are given indicating the anticipated number of printed output lines, the amount of time, or the number of cards, if any, to be punched. If the number of lines and the time are not specified, the computer will assume its own default values and process the job accordingly.

The remaining statements, except the comments beginning with //*, are standard JCL packages that are placed in the open file by using the PLORTS instruction to COPY (not the GENIRAL verb). In order to set up the job in the example, the user at a terminal would have to open a PLORTS file, give it a name, and type the following commands preceded by line numbers:

```
COPY ID.GENIRAS
COPY SETUP.ENGDATA
COPY GENIRAS or // EXEC GENIRAS
the GENIRAL instructions (preceded by line numbers)
COPY DESCRIPT.ENGDATA
//* comments
```

The comment statements, beginning with//*, may be inserted with appropriate line numbers at any point in the program to become stored with the contents of the file. Any line beginning with //* in the first three columns is not interpreted by the computer as an instruction.

The remainder of this section will illustrate the use of the other verbs defined for GENIRAS. Any JCL statements appearing in these examples are taken from those already stored in PLORTS files and available through the PLORTS COPY command. Use of GENIRAS at Level 1 presupposes that no knowledge of JCL is necessary. In the following pages, the verbs defined for GENIRAS are discussed in alphabetical order. Whenever possible, similarities in the usage of different verbs will be mentioned for reference.

ADD

The verb ADD is used to add the corresponding elements of sev l existing data groups to create a new data group. The ADD procedure automatically handles the matching of corresponding elements from the data groups to insure that the new data group is properly calculated from the existing data elements. If an element from any of the existing groups is missing, the corresponding sum element in the new data group will be omitted.

The verb ADD is employed through a GENIRAL statement of the form:

ADD -----, -----,... -----, AND ----- GIVING -----.

Each of the blanks is to be filled with an appropriate data name. As indicated, any number of data names may be included in the first part of the statement after ADD. However, only one data name can follow GIVING. All the data names used, except for the one after GIVING, must refer to existing data.

The new data, created by summing the existing data, becomes stored under the data name entered in the statement after the keyword connector GIVING. This new data is stored only for the duration of the job, and does not become a part of the permanent set of data files. In order to save new data permanently, the methods discussed with the verb SAVE must be employed.

Other verbs in GENIRAL that create new data files in a manner similar to ADD are SUBTRACT, MULTIPLY, and DIVIDE. In each case, the specified operation is performed on the

individual data elements automatically matched from the requested files. So that the matching will be done efficiently, the files names for these operations should be sorted (using the verb SORT) into the same sequence. If index attributes or criteria are specified, the files should be indexed similarly.

The following example shows the use of the verb ADD.

```
ADD EQUIPMENT_COSTS (2250), RESEARCH_EXPENDITURES
    (2250), AND EQUIPMENT_COSTS (2250)
    GIVING PROJECT_COSTS_68_69_70; CRITERION=
    'SPECIAL PROJECTS'.
```

Clearly, the new data group will contain the three-year cost totals for special projects in the Department of Metallurgy and Mining Engineering (code 2250) for the years during which data were recorded for all three variables. In this example, 'SPECIAL PROJECTS,' specified in the options clause, must refer to a criterion defined earlier in the job. No options are defined for ADD.

ANALYZE

The verb ANALYZE is employed through a GENIRAL statement of the form:

ANALYZE ----- USING -----, -----,... -----, AND -----.

The name of the data to be analyzed follows the verb. If no option is specified, a multiple linear regression analysis is performed on these data in terms of the data referred to after USING. The data to be analyzed is treated as the dependent variable, while the others are considered independent.

An alternative use of the verb ANALYZE is associated with GENIRAL statements of the following form:

ANALYZE ----- AGAINST -----; MODEL= -----.

In this application of the verb, the analysis is used for curve fitting. Again, the data name following ANALYZE refers to the dependent variable, while the data name following AGAINST is the independent variable (possibly, an index attribute like YEAR). If other index attributes or criteria apply, they may be included in the statement in the standard manner.

Both usages of the verb ANALYZE permit the specification of the model for the analysis. The variable MODEL, specified in an options clause, may be set equal to LINEAR, QUADRATIC, or EXPONENTIAL. If no options clause is included, GENIRAS assumes MODEL=LINEAR by default. The results of the analysis are printed unless PRINT_OPTION=NO is specified. The results are stored for the remainder of the job if the keyword GIVING is included in the statement and is followed by a data name.

AVERAGE

The verb AVERAGE enables the user to obtain the average of any data group stored in the GENIRAS system. The AVERAGE procedure produces the sum of the elements in a data group, counts the number of numeric and non-numeric data elements, counts the number of zero-valued elements, calculates and prints the average of the numeric data, and calculates the average of the non-zero data.

The verb AVERAGE is employed through a GENIRAL statement of the form:

AVERAGE -----, -----, -----,... -----, AND -----.

For every data group mentioned in the statement, a separate average is calculated and printed unless PRINT_OPTION=NO is placed in an options clause. If PRINT_OPTION=ALL is specified, the additional values for the sum, the number of non-numeric elements, the number of zero-valued elements, and the average of only the non-zero elements will be printed for each data group named in the statement. The average of the numeric elements in a data group (the only value automatically printed if no options are specified), is saved for the remainder of the job under a name of the form: AVG_data_name.

Other verbs that operate in a similar way are SUM and COUNT. As with any GENIRAL statement, index attributes or criteria may be placed in the statement to modify any or all of the data names. The following example illustrates the type of instruction used to cause the calculation of an

average.

AVERAGE GRAD_SALARIES (2250,1970); PRINT_OPTION=NO.

The average of the salaries of 1970 graduates from department 2250 will be calculated. Because the option is specified, no printout will be created at the computer facility, but the data item AVG_GRAD_SALARIES (2250,1970) will be saved for use later in the job.

COUNT

The verb COUNT enables the user to obtain a count of the elements in any data group stored in the computer system. The COUNT procedure totals the number of items in each specified data group and creates a printout of the totals at the computer facility.

The verb COUNT is employed through a GENIRAL statement of the form:

COUNT -----, -----, -----,... -----, AND -----.

If only one data name is given, the AND is not necessary. Each count will be saved for the remainder of the job under a name of the form: COUNT_data_name. If no printout of the counts is desired, PRINT_OPTION=NO should be declared.

Other verbs that operate in a very similar way are AVERAGE and SUM. As with any GENIRAL statement, index attributes or criteria may be placed in the statement to modify any or all of the data names. The following example illustrates the type of instruction used to produce a count.

COUNT BIBLIOG; CRITERION=FOREIGN, PRINT_OPTION=NO.

The number of foreign references in the bibliography of the Documents Center will be calculated as a result of this statement. Because the option is specified, no printout will be created at the computer facility, but the variable COUNT_BIBLIOG will be available for further processing.

DEFINE CRITERION

The DEFINE procedure for criteria applies to previously defined data files. A criterion is defined through a GENIRAL statement of the form:

```
DEFINE CRITERION -----.
*BEGIN (-----)
     ------------
     ------------
*END
```

The blank after CRITERION is filled with the name of the criterion being defined. The name of the criterion also can be entered at the beginning of the data section in parentheses after *BEGIN,

The remainder of the data section contains the statements specifying the criterion. Each of these statements is of the form:

```
name relation constant connector
name relation constant connector
-------------
-------------
```

Each name refers to the name of a particular field. The relation is the desired relation between the values entered in the record fields and some reference constant. The connector in the statement is a comma, AND, or OR.

The following example illustrates the definition of a criterion:

```
DEFINE CRITERION MAJOR_EXP_1961_TO_1965.
*BEGIN (MAJOR_EXP_1961_TO_1965)
     YEAR GE 1961 AND YEAR LE 1965
     EXPENSES GE 1000
*END
```

EXPENSES is the name of the existing data file to which this criterion is to be applied. The lower limit on the expenses of interest is $1,000. Only expenses incurred during the years 1961-1965 will be examined.

The following permissible relations are shown with all their equivalent forms.

'equal to' can be expressed as: EQUAL TO
EQ
=

'not equal to' can be expressed as: NOT EQUAL
NE
¬

'greater than' can be expressed as: GREATER THAN
GT
>

'less than' can be expressed as: LESS THAN
LT
<

'greater than or equal to' can be exptessed as: GE
>=

'less than or equal to' can be expressed as: LE
<=

In specifications of limits involving both AND and OR, the AND relationships will be checked first, and then the OR relationships will be checked. For example, the specification

```
YEAR GE 1955 AND YEAR LE 1960 OR
YEAR GE 1965 AND YEAR LE 1970
```

identifies data from the years 1955 to 1960 or from 1965 to 1970.

DEFINE NAMES (used for data identification)

One application of the verb DEFINE is to identify the source of data items. As described in the section of this manual dealing with data identifiers, a data name can be identified with a particular file and subfile by using dataname:filename:subfilename whenever the data name has to be used. The DEFINE NAMES procedure can be used as a more convenient means of associating data items with particular files and subfiles. The procedure is employed through a statement of the form:

DEFINE NAMES FROM filename:subfilename.

A subfile name is included in the statement only if it is needed for locating the desired data items.

Once the DEFINE NAMES procedure has been specified to the computer during the execution of a job, all the names of data items mentioned in the job will be located automatically by the appropriate file name and subfile name. More than one file name and one subfile name can be declared using the DEFINE NAMES procedure only if the data items to be mentioned in the job are unique to the specified files and subfiles.

The following example illustrates the use of this verb:

```
DEFINE NAMES FROM ADMIN_FILES:EXPENSES AND RECORDS:
    ENROLLMENT.
SUM SALARIES_AND_WAGES FOR (2250,1970); CRIT=ACADEMIC.
DIVIDE SUM_SALARIES_AND_WAGES BY UNDERGRAD_ENROLLMENT
    GIVING TEACHING_COST_PER_STUDENT FOR (2250,1970).
```

The data item SALARIES_AND_WAGES is obviously from the EXPENSES subfile of the ADMIN_FILES file. UNDERGRAD_ENROLLMENT, on the other hand, is from the ENROLLMENT subfile of the RECORDS file. In the statements following the DEFINE NAMES procedure, and in most of the statements found elsewhere in this manual, the file names and subfile names have been omitted because they are assumed to be defined in this manner.

Another application of the verb DEFINE is to describe the structure of a particular data file. The structure of such a file can be declared from a keyboard terminal by using a statement of the following form:

```
DEFINE DATA FROM INPUT GIVING NAMES:-----;
    FORM='-----'.
*BEGIN
    FILE_NAME='-----'
    ---------------
    ---------------
*END
```

The word INPUT in the first line identifies the terminal as the source of the definition statements.

It is obvious from the previous example that NAMES is used as the master prefix for all the file names created by a user. The name of the particular file to be defined is included in the DEFINE statement after NAMES: and in the blank after FILE_NAME in the data section. The FORM referred to in the DEFINE statement is the form used for the

information in the data section.

Three different forms for the data section are acceptable. If the definition information is given in the form of a COBOL file description, FORM='COBOL' should be declared in the options clause. If the information is in the form of PL/I declare statements, then FORM='PLI' should be stated. If the information is already in the fixed GENIRAL format, FORM='FIXED' can be declared, although FORM='FIXED' is the default value if the options clause is omitted. Finally, if the information is in the GENIRAL free-form data description language, FORM='GENDDL' should be declared.

The FORM declaration in the options clause tells the system which syntax file and translator are to be used to convert the data section into a GENIRAL fixed-form file description. The GENIRAL free-form data description language (GENDDL) has the following structure:

```
FILE IS filename
SUBFILE IS subfilename
FIRST_INDEX IS indexname (field_starting_position, type,
     length, decimal_position)
SECOND_INDEX IS indexname (field_starting_position, type,
     length, decimal_position)
dataname IS (field_starting_position, type, length,
     decimal_position)
dataname IS (field_starting_position, type, length,
     decimal_position)
-----------
-----------
```

(Anywhere in the above description, = can be substituted for IS.)

In the example, dataname is the name of the data being defined. The filename is the name chosen for the file; subfilename is the name chosen for the subfile. The index attributes for the data item also are named and defined in this data section. Lastly, the fields in the records are specified.

The field descriptions contain the following information: the location of the first column in the field; the type of field; the length of the field; and the number of decimal positions from the right edge of the field.

Two types of field can be defined. Entering A, for alphanumeric, in the field description means that the field is to contain a word or words composed of alphanumeric characters. Entering N, for numeric, means that the field is to contain a number. The length of the field is declared in the statement after the type of field is declared. Lastly, the number of decimal places reserved to the right of the decimal point is declared. (The default value is zero.)

The following example illustrates a file description:

```
     DEFINE DATA FROM INPUT GIVING NAMES:EXPENSES;
         FORM='GENDDL'.
*BEGIN
     FILE IS EXPENSES
     FIRST_INDEX IS DEPARTMENT (1,A,4)
     SECOND_INDEX IS YEAR (5,N,4)
     SALARIES IS (9,n,10,2)
     --------------
     --------------
     --------------
*END
```

DEFINE NAMES (used for data identification--page 5)

The index attributes are assumed to be DEPARTMENT and YEAR for this application. The DEPARTMENT field begins in column 1, is alphabetic, and has no decimal places. The YEAR field begins in column 5, is numeric, and has no decimal places. The SALARIES themselves are in a numeric field, which is 10 places long, beginning in column 9. The SALARIES field has an assumed decimal point which allows two decimal places in the number. The following variations are all equivalent to the field description for SALARIES:

(9,N,10,2); (9,N,(10,2)); or (9,N (10,2)).

The GENIRAL free-form data description language permits a great deal of flexibility in the use of punctuation and in the order of the statements. The items can be defined in any order in the data section, and the field description can be adapted to a wide variety of forms.

DEFINE PROCEDURE

The verb DEFINE can be employed to outline Level-1 procedures that may be required for data processing. (Level-2 procedures are defined using the COMPILE verb.) A Level-1 procedure is defined through a GENIRAL statement of the form:

```
DEFINE PROCEDURE (-----) USING -----, -----,...
    AND -----.
*BEGIN (-----)
    --------------
    --------------
*END
```

The name of the procedure appears in the parenthesis after PROCEDURE (interchangeable with PROC) and again, in the data section, in the parentheses after *BEGIN. Dummy names may be placed in the DEFINE statement after USING to show how item arguments for the procedure will be handled.

The following example illustrates the definition of a procedure:

```
DEFINE PROCEDURE ('PERCENTAGE LIST') USING A AND
    B.
*BEGIN ('PERCENTAGE LIST')
    ADD A TO B  GIVING C.
    DIVIDE C BY A GIVING RATIO.
    MULTIPLY RATIO BY 100 GIVING PERCENT.
    LIST A, B, C, AND PERCENT.
*END
```

The dummy variables of the procedure are A and B. The name of the procedure is enclosed in single quotation marks because of the space between the words. The data item

identified by C is the sum of the items represented by A and B.

The following example illustrates the use of this procedure after it has been defined:

```
PERFORM 'PERCENTAGE LIST' USING EXPENSES(1970) AND
    EQUIPMENT_COST(1970).
```

The system will refer to the stored definition of the procedure and perform it with EXPENSES(1970) substituted for A and EQUIPMENT_COST(1970) substituted for B. If EXPENSES and EQUIPMENT_COST are transposed in the PERFORM statement, EQUIPMENT_COST(1970) will be substituted for A and EXPENSES(1970) will be substituted for B. Consequently, the resulting calculations will be different. The order of the data names in the PERFORM statement must follow the order of the dummy variables in the definition.

DIVIDE

The verb DIVIDE is used to divide the corresponding elements of two existing data groups to create a new data group. The DIVIDE procedure automatically handles the matching of corresponding elements from the data groups to insure that the new data group is properly calculated from the existing data elements. If an element from either of the existing groups is missing, the corresponding division result will be omitted in the new data group.

The verb DIVIDE is employed through a GENIRAL statement of the form:

DIVIDE ----- BY ----- GIVING -----.

The three blanks are to be filled with the appropriate data names. The first two blanks must be filled with the names of existing data groups. The name after GIVING will be the name of the new data group created by matching and dividing the items in the first two groups.

The new data created will be stored only until the end of the job, and does not become a part of the permanent set of data files. In order to save this data, the methods discussed with the verb SAVE must be employed.

Other verbs in GENIRAL that create new data files in a manner similar to DIVIDE are ADD, SUBTRACT, and MULTIPLY. In each case, the specified operation is performed on the individual data elements automatically matched from the requested files. Since matching occurs, the files named for these operations should be sorted (using the verb SORT)

into the same sequence. If index attributes or criteria are specified, the files should be indexed similarly.

The following example shows the use of the verb DIVIDE.

```
DIVIDE OFFICE_SPACE BY TOTAL_SPACE GIVING
OFFICE SPACE/SPACE' FOR (*,1970).
```

Using 1970 data, the ratio of office space to total space will be calculated for every department (assuming department was defined as the first index of OFFICE_SPACE and TOTAL_SPACE).

ENTER

The verb ENTER is used to enter a file of data into the system (usually from a keyboard terminal). This verb is employed through a GENIRAL statement of the form:

```
ENTER -----USING -----(-----).
*BEGIN (the asterisks for *BEGIN and *END must be in
    column 1)
------------
------------
*END
```

The blank after ENTER is filled with the name to be assigned to the newly entered file. The blank after USING is filled with the name of the file containing the syntax information for interpreting the data. The blank in parentheses is for the name of the routine that converts the format of the incoming data (described by the syntax information), into an alternate format. The data, in the form to be translated, follows the statement in the data section.

Translators already exist for common forms of input data. If the input data is in the form of COBOL file descriptions, the syntax file COBOLFD should be used. If the input data is in the form of PL/1 declare statements, the syntax file PL1DCL should be used. If the input data is in the GENIRAL free-form data description language, the syntax file called GENDDL should be used. Lastly, data in the form of transactions can be entered using the TRANLAN syntax.

Translator programs and syntax descriptions for other applications should be written only by an experienced

Level-2 programmer who is familiar with GENIRAS.

The following example illustrates the use of the verb ENTER:

```
ENTER TRANSACTIONS USING TRANLAN(TRANTAN).
*BEGIN
ACC#7446, DEBIT,PØ3091, 'ACME CHEMICALS'
*END
```

The first transaction described above involves a purchase from Acme Chemicals made against account number 7446. This transaction involves a debit to cover purchase order number 3091. Although only one transaction is being entered in the example, any number of transactions could be included in the data section.

The DEFINE and ENTER procedures are interrelated in GENIRAL in that either procedure may be used first when creating a data file. The data can be entered into the system using the ENTER procedure, and then the form for the data file can be defined using the DEFINE procedure or vice versa. The DEFINE procedure is used to outline the new file structure with subfiles, index attributes, and the appropriate field formats for its records; the Enter procedure is used to enter and store data under a particular file name.

GRAPH

The verb GRAPH has many options to permit the plotting of point or bar graphs in a wide variety of formats. Up to five dependent variables can be plotted against a single independent variable.

The verb GRAPH is employed through a GENIRAL statement of the form:

GRAPH -----, -----, -----, -----, AND ----- AGAINST -----.

The blanks in the statement after the word GRAPH are filled with the names of the data groups to be plotted against the single control group (named after AGAINST). (The control group is often a common index attribute of the dependent variables). The matching of corresponding elements is accomplished automatically by the GRAPH procedure.

The following example illustrates the use of the verb GRAPH:

```
GRAPH BS_DEGREES, MS_DEGREES, AND PHD_DEGREES AGAINST
     YEAR FOR DEPARTMENT (2250); FORM='VERTICAL BAR'.
```

This statement will create a graph with a set of three bars plotted vertically over each value of YEAR. The bars in each set will represent the three dependent variables in the order that they are mentioned in the statement. The GRAPH routine will automatically select a different character for each dependent variable, and will list the characters with their corresponding variable names on the printout.

In most cases, a simple statement naming the independent and dependent variables is all that is necessary to

create a graph from GENIRAS data. In other cases, certain options may be helpful. The numerous GRAPH options have been tabulated below with their default values.

VERTICAL_LABEL='-----' indicates the label to be printed for the vertical axis. Nothing is printed as the label if this option is not specified.

HORIZONTAL_LABEL='-----' indicates the name to be printed for the horizontal axis. Nothing is printed as the label if this option is not specified.

TITLE='-----' indicates the title to be printed on the graph. Nothing is printed as the title if this option is not specified.

FORM='VERTICAL BAR' or 'VERTICAL POINT' or 'HORIZONTAL BAR' or 'HORIZONTAL POINT' indicates the direction and display method of the graph. The direction (horizontal or vertical) is the direction of the axis of the dependent variables. The display method (POINT or BAR) indicates whether a single point or a bar is plotted above the axis of the independent variable. The default values, if the option is not specified, are VERTICAL and POINT.

The following options are for special applications, and normally are not necessary when plotting a graph.

GRAPH_SIZE='---' indicates the width of the entire graph. The maximum value for the width is 100 columns. The default value of this option also is 100.

AUTOMATIC_ADJUSTMENT='-----' indicates that the width of the graph can be automatically adjusted by the system if the space required to plot the desired points exceeds the specified or default value of the width. The default value of this option is YES.

NUMBER_OF_INTERVALS='---' indicates the number of intervals (for one dependent variable, the number of points) included in the graph. The value given for the NUMBER_OF_INTERVALS, times either the specified or the default value of the INTERVAL_SIZE, fixes the width of the graph. The default value of this option is 20, and the system will adjust the width of the graph accordingly, despite the fact that less than 20 intervals may be needed.

INTERVAL_SIZE='---' indicates the number of columns between the plotted points of a particular variable. For example, if a bar graph is plotted with bars 1 column wide, then the distance between successive bars will be the INTERVAL_SIZE. In general, the number of spaces left between successive points of a particular plotted variable will be the INTERVAL_SIZE minus the WIDTH. The default value of this option is 5.

INTERVAL_SPACING='---' indicates the number of spaces to be left between successively plotted groups of points. For example, if four data groups are plotted on a bar graph against year, there will be four bars associated with each value along the year axis. The INTERVAL_SPACING is the distance between the last bar in one group and the first bar in the next group. Specifying the INTERVAL_SPACING guarantees a certain minimum amount of space between the groups for data points for several dependent variables. The default for the INTERVAL_SPACING is 1.

POINT_SPACING='-----' indicates the spacing, along the axis of the independent variable, between the points plotted for the values of the dependent variables corresponding to a single value of the independent variable. For example, if four dependent variables are plotted against the values of the independent variable, a POINT_SPACING of 1 will put the points for the dependent variables in successive columns. The default value of this option is, which puts all the points in one column. If the points to be graphed will be very similar in value, it is a good idea to separate them using this option because the printer cannot plot two points in the same space.

WIDTH='---' indicates the width of the bars or points to be plotted. The default value is 1.

STARTING_POSITION='---' indicates how far from the left or bottom edge of the graph the first point is to be plotted. The default value is 2.

STARTING_POINT='---,---,... ---' indicates how far from the left or bottom edge of the graph successive points are to be plotted against the first value of the independent variable. The first number in the option specification gives the distance of the first point from the edge, the second number gives the distance of the second point from the first point, and so on for up to five dependent variables. The default values for up to five dependent variables are 2,2,2,2 and 2. If the graph has only one dependent variable, the STARTING_POINT and STARTING_POSITION options are

equivalent.

RANGE_MAX='---' indicates the maximum value to be assigned along the axis of the dependent variables. The default for this option is that the system finds its own 'natural' maximum which just exceeds the maximum value of all the dependent variables being plotted. Efficient scaling of the graph depends on the maximum and minimum values of the dependent variables and not on some preset limits. It is usually not a good idea to specify these limits in advance. The system will automatically scale the graph into equal, 'natural' intervals that include the maximum value if this option is omitted.

RANGE_MIN='---' indicates the minimum value to be assigned along the axis of the dependent variables. The default for this option is a 'natural' minimum which is just less than the minimum value of the dependent variables being plotted. If RANGE_MIN and RANGE_MAX are specified, the values must give enough range to plot all the points of the graph. If the range is not sufficient, any points to be graphed outside the range will be omitted.

LIST

The LIST procedure creates a list, in tabular form, of the elements from data groups with matching pairs of index attributes. If no index attributes are specified for the matching process, the index attributes of the first data group will be used by default.

The verb LIST is employed through a GENIRAL statement of the form:

LIST -----, -----, -----,... -----, AND -----.

The data groups named in the statement are not changed in any way, but are merely listed.

If the list is to be broken into groups on the basis of a particular index attribute or criterion, it should be mentioned in the options clause as a CONTROL. For example, the statement

LIST SALARIES_AND_WAGES FOR DEPT (2250); CONTROL=YEAR.

will give the desired listing broken into yearly groups. If no method of breakdown is specified, the data is printed in a continuous list. If a breakdown is desired, the data groups first should be sorted using the verb SORT into the sequence of the CONTROL item.

MULTIPLY

The verb MULTIPLY is used to multiply the corresponding elements of several existing data groups to create a new data group. The MULTIPLY procedure automatically handles the matching of corresponding elements from the data groups to insure that the new data group is properly calculated from the existing data elements. If an element from any of the existing groups is missing, the corresponding element in the new data group will be omitted.

The verb MULTIPLY is employed through a GENIRAL statement of the form:

MULTIPLY -----, -----, -----,... -----, AND -----
GIVING -----.

Each of the blanks is to be filled with an appropriate data name. As indicated, any number of data names may be included in the first part of the statement after MULTIPLY. However, only one data name can follow GIVING. All the data names used, except for the one after GIVING, must refer to existing data.

The new data, created by multiplying the existing data, becomes stored under the data name entered in the statement after the keyword connector GIVING. This new data is stored only for the duration of the job, and does not become a part of the permanent set of data files. In order to save new data permanently, the methods discussed with the verb SAVE must be employed.

Other verbs in GENIRAL that create new data files in a manner similar to MULTIPLY are ADD, SUBTRACT, and DIVIDE.

In each case, the specified operation is performed on the individual data elements automatically matched form the requested files. Since matching occurs, the files named for these operations should be sorted (using the verb SORT) into the same sequence. If index attributes or criteria are specified, the files should be indexed identically.

The following example shows the use of the verb MULTIPLY.

```
MULTIPLY WORK_HOURS BY AVG_WAGE_RATE GIVING
   EST_LABOR_COSTS FOR DEPT (2250); CRIT=MAINTENANCE.
```

Using an average figure for the wage rate, a value of work hours (possibly projected) is multiplied to give an estimate of maintenance labor costs for department 2250. It is important to note that the basic form for this statement differs somewhat from the general form given earlier. The only difference is that the connector BY is a more natural English usage than AND in the situation where only two data groups are multiplied.

PERFORM

The verb PERFORM is used to execute a specific prewritten Level-1 procedure stored in GENIRAS. The options applying to PERFORM depend entirely on the options defined for the procedure being called. The definition of Level-1 procedures is discussed under the verb DEFINE.

The verb PERFORM is employed through a GENIRAL statement of the form:

PERFORM ----- USING -----, -----,... AND ----- GIVING -----.

The first blank in the statement contains the name of the prewritten procedure. If the procedure has been defined to use arbitrary data groups, the data names for a specific application must be included in the PERFORM statement with the connector USING. If a new data group is created by the procedure being called, the name for the new group must follow the connector GIVING in the statement.

The following example illustrates the use of the verb PERFORM:

PERFORM TREND_ANALYSIS USING ENROLLMENT FOR CURRIC (2250).

This statement assumes the existence of a prewritten, Level-1 procedure called TREND_ANALYSIS. The analysis apparently has been written to operate on whichever data group is specified in the statement after USING. In this case, the data group is ENROLLMENT, where curriculum and year have been defined as the index attributes of enrollment data.

PRINT

The verb PRINT can be used in place of the verb EXECUTE whenever a printed report results from the execution of a prewritten procedure. The details of the application of the verb PRINT are the same as for the verb EXECUTE, where they are described.

The following example shows the use of the verb PRINT instead of EXECUTE to make a statement read in a more natural manner without changing its function:

```
PRINT TRANSACTION_REPORT FOR DEPT (2250).
```

This statement reads more naturally than its equivalent form using EXECUTE:

```
EXECUTE TRANSACTION_REPORT FOR DEPT (2250).
```

PROJECT

The PROJECT verb provides a simple means for obtaining approximate values of future data on the basis of past trends. The verb PROJECT is employed through a GENIRAL statement of the form:

```
PROJECT ----- FROM ----- TO -----; MODEL='-----',
    option_name='-----',...option_name='-----'.
```

The first blank is filled with the name of the data group which is to be projected. After the keyword FROM, the starting point is given in the form of a single-valued item or constant. After the keyword, TO, the stopping point is given.

The following example illustrates the use of the verb PORJECT:

```
PROJECT EXPENSES FROM YEAR (1970) TO YEAR (1985);
    MODEL='LINEAR', SLOPE='1.10', starting_value
    36000000.
```

In this particular example, EXPENSES are being projected through 1985. The starting point is given as the single value of EXPENSES corresponding to the year 1970. The stopping point is also single-valued and refers to the value calculated for 1985. The mathematical MODEL chosen for the projection is to be LINEAR, and the SLOPE to be used for the projected data is 1.10. For a linear model, the system must have access to a value for the SLOPE. Other types of models require different parameters identified in the options clause.

If the data group to be projected already has been analyzed (using the ANALYZE procedure) the parameters nec-

essary to make a projection will be available during the remainder of the job.

The three possible models for analysis and projection are LINEAR, QUADRATIC, and EXPONENTIAL. As the result of a LINEAR analysis, the value of the SLOPE will be stored under SLOPE_data_name, where data_name is the name of the item that was analyzed. If a QUADRATIC model is used in an analysis, the coefficient of the quadratic term will be stored under QUAD_data_name; the coefficient of the linear term will be stored under SLOPE_data_name. Finally, if an EXPONENTIAL analysis is performed, the EXPONENT will be stored under EXP_data_name. These same parameters are required with the appropriate model when a projection is being made. If an analysis has been performed using the same type of model as the model to be used for projection, these parameters may be identified in the options clause by name. Otherwise, numerical values must be given.

The following examples show the PROJECT verb being used with and without a preceding analysis:

```
ANALYZE ENROLLMENT AGAINST YEAR FOR DEPT(2250);
     MODEL=EXPONENTIAL.

PROJECT ENROLLMENT(2250) FROM YEAR(2250,1970)
     TO (2250,1985); MODEL=EXPONENTIAL, EXPONENT=EXP_
     ENROLLMENT(2250), STARTING_VALUE=84.
```

or

```
PROJECT ENROLLMENT(2250) FROM YEAR(2250,1970)
     TO (2250,1985); MODEL=EXPONENTIAL, EXPONENT='1.17',
     STARTING_VALUE=84.
```

The first example shows the use of the PROJECT procedure after an analysis using the same type of model. The independent variable for both the analysis and the projection is YEAR, and the starting point is the single value of the enrollment in department 2250 during 1970. The stopping point is the year 1985. The data name may or may not be repeated when the stopping point is identified. In this example, it is not. Because the analysis precedes the projection, the exponent is available by name. The second example performs a projection between the same endpoints without depending on a preceding analysis. The exponent has been calculated by some other method and has been given as a numerical value in the options clause.

RESTORE

Use of the verb RESTORE follows the application of the verb SAVE. If data items have been created during the execution of a previous job and have been saved using a statement of the form:

SAVE -----, -----,... AND -----.,

they may be recalled for use in subsequent jobs by including a statement of the form:

RESTORE -----, -----,... AND -----.

The blanks are filled with the names of the data items that are being retrieved from storage.

If the data items have been stored in a file created by the user, they are retrieved from that particular file by a GENIRAL statement of the form:

RESTORE -----, -----,... AND ----- FROM -----.

The name of the file created by the user follows FROM in the statement. Referral of the system to particular file requires the inclusion of appropriate JCL device description statements with the job instructions.

SAVE

The SAVE procedure is the standard method used to store newly created data items. New data items result from the arithmetic verbs like ADD, SUM, and MULTIPLY and from certain other item-oriented verbs in GENIRAL. The SAVE procedure is applicable only to new data items, and not to new data files (like those created by the GENIRAL verbs SORT and MERGE). The verb SAVE is employed through a GENIRAL statement of the form:

SAVE -----, -----,... AND -----.

If data items have been saved from a previous job, and if they must be located and made available in a later job, the verb RESTORE must be employed in a statement of the form:

RESTORE -----, -----,... AND -----.

The blanks are filled with the names of the required data items.

If the user wishes to place a data item in some file other than the one automatically reserved for item storage, the SAVE statement should be modified to read:

SAVE -----, -----,... AND ----- GIVING -----.

The last blank is filled with the name of the file in which the data items will be stored. The file named after GIVING (interchangeable with ONTO) must be a partitioned data set described by appropriate device description statements (DD statements in JCL).

The following example illustrates the use of the verb SAVE:

```
SAVE SUM_EXPENSES ONTO MYFILE.
```

MYFILE has been described elsewhere in the job instructions as a partitioned data set that is to be kept after the termination of the job. The item being saved is a group of data items formed by adding items from the EXPENSES file. This group can be restored for use in a later job with the following statement:

```
RESTORE SUM_EXPENSES FROM MYFILE.
```

SELECT

SELECT is a Level-2 procedure which provides an efficient means for selecting a subset of records from a data file. The SELECT procedure performs the same function as the verb COPY used with a criterion, but SELECT is more efficient. Unlike COPY, the procedure for selecting data items must be defined for each application of the verb SELECT and is not limited to previously defined criteria.

The criteria used in the SELECT procedure are described in a Level-2 program using the COMPILE procedure. Specification of the SELECT feature takes the following form:

```
SELECT:PROCEDURE:
%INCLUDE DESCRIPT(filename);
%INCLUDE PL1MACRO(SELECT1);
IF ------------
%INCLUDE PL1MACRO(SELECT2);
```

The filename in the specification is the name of the file to which the SELECT procedure is to be applied. The blank after IF in the specification is filled with an appropriate form of a PL/1 IF statement (<u>without</u> a THEN or an ELSE clause).

If the SELECT procedure is entered at the terminal, it can be compiled by entering the lines COMPILE SELECT. and *BEGIN (with the asterisk in column 1) before the first line of the specification and by entering *END after the last line.

The following example illustrates the compilation of a SELECT procedure entered at a terminal:

```
     COMPILE
*BEGIN
     SELECT:PROCEDURE;
      %INCLUDE PL1MACRO(SELECT1);
      %INCLUDE DESCRIPT(CUMFILE);
      IF CLASS=01 AND ENDING_HOURS>40
      %INCLUDE PL1MACRO(SELECT2);
*END
```

This SELECT procedure is to be used on a file called CUMFILE to select records on the basis of the CLASS code and the number of credit hours at the end of the term. The desired records from CUMFILE are selected and placed in a new file called WORKFILE using the following statement:

```
SELECT WORKFILE FROM CUMFILE.
```

The SELECT procedure must be recompiled each time it is needed, and it can be changed at will. (WORKFILE is a special temporary file, stored only until the end of the job, and it requires no accompnying JCL device description statements. Any other name for a temporary or permanent file requires such a device description.)

SORT

The verb SORT can be used to reorder the records in a data file before the records are processed. Any of the procedures requiring matching of the corresponding elements from different data files should be preceded by a SORT statement putting the records into the proper order in their respective files.

The verb SORT is employed through a GENIRAL statement of the form:

SORT ----- ONTO -----; CONTROL='-----',SIZE='-----'.

The first blank in the statement is filled with the name of the file to be sorted. The connector ONTO (interchangeable with the connector GIVING) precedes the name for the new, sorted file. The following example illustrates the use of the verb SORT:

```
SORT CUMFILE ONTO WORKFILE; CONTROL=SOC_SEC_NO,
    SIZE=E3000.
```

CUMFILE is to be sorted onto a temporary file called WORKFILE. Unlike other temporary or permanent files, WORKFILE requires no device description (DD) statement, written in JCL and included with the job instructions. The necessary DD statements for WORKFILE are entered automatically when the job is executed. The use of any other file name after the word ONTO requires an appropriate DD statement.

The order of the records in the sorted file (WORKFILE in the example) depends on the CONTROL option which must be

specified in the options clause. The form of the CONTROL option is: 'ITEM_NAME,ITEM_NAME,ITEM_NAME,...ITEM_NAME'. In the example given above, the records in CUMFILE will be sorted by social security number. The word SOC_SEC_NO can be used as the CONTROL item in this case because it has been defined as the name of the field for the social security number in CUMFILE records. If no name had been assigned to the field for the social security number, a pair of numbers, separated by a comma and indicating the starting column and column length of the field, could be indicated as the CONTROL_NUMBERS. If a multiple CONTROL like 'YEAR,NAME' had been used, the records would have been sorted in ascending order by year and then alphabetically by name within each year. The ascending order of the years, from the earliest to the most recent, and the ascending alphabetical order, from A to Z, can be reversed by placing (D), for descending, after the appropriate CONTROL items.

Because peripheral storage space will be needed to hold the records while they are being sorted, the total SIZE, in terms of the number of records, must be specified. For example, the size in the example is written as 'E3000" because the estimated total number of records is slightly less than 3,000. (The estimate of the SIZE should never be too low.)

An option which has not been specified in the example is RECORD_LENGTH. If the length of the records in the

file is 80 columns, this option is not mandatory because its default value is 80. For longer or shorter records, however, the RECORD_LENGTH must be specified in the options clause.

SUBTRACT

The verb SUBTRACT is used to subtract the corresponding elements of two existing data groups to create a new data group. The SUBTRACT procedure automatically handles the matching of corresponding elements from the data groups to insure that the new data group is properly calculated from the existing data elements. If an element from either of the existing groups is missing, the corresponding subtraction result will be omitted in the new data group.

The verb SUBTRACT is employed through a GENIRAL statement of the form:

SUBTRACT ----- FROM ----- GIVING -----.

The three blanks are to be filled with the appropriate data names. The first two blanks must be filled with the names of existing data groups. The name after GIVING will be the name of the new data group created by matching and subtracting the items in the first two groups.

The new data created will be stored only until the end of the job, and does not become a part of the permanent set of data files. In order to save this data permanently, the methods discussed with the verb SAVE must be employed.

Other verbs in GENIRAL that create new data files in a manner similar to SUBTRACT are ADD, MULTIPLY, and DIVIDE. In each case, the specified operation is performed on the individual data elements automatically matched from the requested files. Since matching occurs, the files named for these operations should be sorted (using the verb SORT) into

the same sequence. If index attributes or criteria are specified, the files should be indexed similarly.

The following example shows the use of the verb SUBTRACT.

```
SUBTRACT ELECTRICITY_COSTS FROM MAINTENANCE_COSTS GIVING
    ADJUSTED_COSTS FOR (*,1970).
```

As a result of this instruction, the adjusted maintenance costs for the year 1970 will be calculated for all departments. In this example, it is important to note the positional nature of the index attribute. Since YEAR is assumed to be the second attribute, the asterisk must be included in the parentheses. If YEAR were the first attribute, it could be written as either (1970,*) or (1970).

SUM

The verb SUM enables the user to obtain the sum of the elements in any data group stored in the GENIRAS system. The SUM procedure adds together all the items in each specified group and creates a printout of the totals at the computer facility.

The verb SUM is employed through a GENIRAL statement of the form:

SUM -----, -----,... -----, AND -----.

If only one data name is given, the AND is not necessary. Each sum will be saved for the remainder of the job under a name of the form: SUM_data_name. If no printout of the sums is desired, PRINT_OPTION=NO should be declared.

Other verbs that operate in a similar way are AVERAGE and COUNT. As with any GENIRAL statement, index attributes or criteria may be placed in the statement to modify any or all of the data names. The following example illustrates the type of instruction used to produce a sum.

SUM MAINTENANCE_COSTS FOR (*,1970); PRINT_OPTION=NO.

The total amount spent during 1970 on maintenance by all departments will be calculated. Because the option is specified, no printout will be created at the computer facility.

UPDATE

The UPDATE procedure is extremely important for keeping files of stored data current and accurate. The UPDATE procedure is called with a GENIRAL statement of the form:

UPDATE -----(---,---) ONTO ----- USING -----(---,---).

The file to be updated is named after the verb UPDATE. In the parentheses following the file name are two numbers indicating, in order, the beginning and the length of the field to be used for matching corresponding records. The name of the new file created by the UPDATE procedure is placed in the statement after the word ONTO. The name of the file containing the information used for updating is listed after USING. Again, the fields used for matching are described by the numbers in parentheses.

The following example illustrates the use of the verb UPDATE:

```
UPDATE CUMFILE(1,20) ONTO NEWCUMFILE USING NEWFILE
     (1,20).
```

The records in CUMFILE are to be updated in the example. The records in CUMFILE are to be matched against those in NEWFILE on the basis of the information contained in the specified field. In this example, the field starts in column 1 and is 20 columns long; it might contain the name of the person to whom the record applies.

As the UPDATE procedure matches records on the basis of the specified field informatin, it has various operations

which it can perform. If there is no element in the old file corresponding to a particular record in the input file, (the record in the input file is new) and if OPTIONS='INSERT' has been specified in an options clause, the new record will be inserted into the updated file. If a record in the old file and its corresponding record in the updating file have been matched on the basis of the specified field, and if the option REPLACE has been specified, the old record will be replaced by the new record in the updated file. Declaring no options in the update statement causes both the INSERT and REPLACE options to be declared automatically.

If OPTIONS='DELETE' is placed in the options clause, the field information will be compared as before, but if the matched records are identical, the record in the old file will be deleted. The INSERT, REPLACE, and DELETE options can be declared individually, as described, or in pairs of the form: OPTIONS='INSERT AND DELETE'. However, REPLACE and DELETE cannot be specified at the same time.

It is important to remember that the format of the records in the updating file and the format of the records in the old file must be identical because entire records, not just field entries, are replaced, inserted, or deleted during updating. For example, if the social security number has been defined as a field of each record (in columns 1-9 of CUMFILE), the social security number would be matched in an UPDATE statement of the form:

```
UPDATE CUMFILE(1,9) ONTO NEWCUMFILE USING NEWFILE (1,9).
```

APPENDICES

APPENDIX A -- GENIRAL SYNTAX

The following table illustrates the complete structural organization of GENIRAL. From the definition of a program as a series of statements, to the specification of the letters for making words, the formation of GENIRAL statements is described in this self-contained syntax. Each 'part of speech' of the language is defined in terms its more basic building blocks. If alternate forms for the composition of a part of speech exist, they are listed. If the formation of a part of speech requires the iterative use of its building blocks, this condition is indicated by repeating the original part of speech in the definition of its composition.

Each part of speech of GENIRAL is enclosed in brackets when it is mentioned in the table. Although the brackets do not appear in actual GENIRAL statements, all other punctuation indicated in the table is required. In addition to the brackets, the connective elements, 'is defined as' and 'or,' are not part of GENIRAL, but are part of the organizational structure of the table.

Syntax Table for GENIRAL

```
[PROGRAM] is defined as [SERIES OF STATEMENTS]

[SERIES OF STATEMENTS] is defined as [STATEMENT] or
                        [STATEMENT][SERIES OF STATEMENTS]

[STATEMENT] is defined as [MAIN CLAUSE]. or
           [MAIN CLAUSE];[OPTIONS CLAUSE]. or
           [MAIN CLAUSE].[DATA SECTION] or
           [MAIN CLAUSE];[OPTIONS CLAUSE].[DATA SECTION]

[MAIN CLAUSE] is defined as [VERB][IDENTIFIER] or
                            [VERB][IDENTIFIER][OTHER TERMS]

[OTHER TERMS] is defined as [CONNECTOR][IDENTIFIER] or
                        [CONNECTOR][IDENTIFIER][OTHER TERMS]

[VERB] is defined as [WORD]

[IDENTIFIER] is defined as [DATA NAME] or
                           [DATA NAME][QUALIFIERS] or
                           [NUMBER]

[DATA NAME] is defined as [WORD]

[QUALIFIERS] is defined as [FILE QUALIFIERS] or
                    [SUBSET QUALIFIERS] or
                    [FILE QUALIFIERS][SUBSET QUALIFIERS] or
                    [SUBSET QUALIFIERS][FILE QUALIFIERS]

[FILE QUALIFIERS] is defined as :[FILE NAME] or
                          :[FILE NAME]:[SUBFILE QUALIFIERS]

[SUBFILE QUALIFIERS] is defined as :[SUBFILE NAME] or
                       :[SUBFILE NAME]:[SUBFILE QUALIFIERS]

[FILE NAME] is defined as [WORD]

[SUBFILE NAME] is defined as [WORD]

[SUBSET QUALIFIERS] is defined as ([INDICES]) or
                           ([CRITERION TERM]) or
                           ([INDICES],[CRITERION TERM])

[INDICES] is defined as [INDEX] or
                        [INDEX],[INDICES]

[INDEX] is defined as [WORD]

[CRITERION TERM] is defined as [CRITERION KEYWORD]=
                               [CRITERION NAME]
```

```
[CRITERION KEYWORD] is defined as CRITERION or
                                  CRIT

[CRITERION NAME] is defined as [WORD]

[CONNECTOR] is defined as [PASSIVE CONNECTOR] or
                          [KEYWORD CONNECTOR]

[PASSIVE CONNECTOR] is defined as AND or
                                  , or
                                  , AND or
                                  TO or
                                  BY or
                                  WITH

[KEYWORD CONNECTOR] is defined as OR or
                                  FROM or
                                  USING or
                                  AGAINST or
                                  MATCHING or
                                  INTO or
                                  ONTO or
                                  ON or
                                  GIVING or
                                  FOR
[OPTIONS CLAUSE] is defined as [OPTION SPECIFICATIONS]

[OPTION SPECIFICATIONS] is defined as [OPTION TERM] or
                    [OPTION TERM][OPTION SPECIFICATIONS] or
                    [OPTION TERM],[OPTION SPECIFICATIONS]

[OPTION TERM] is defined as [OPTION NAME]=[OPTION VALUE]

[OPTION NAME] is defined as [WORD]

[OPTION VALUE] is defined as [WORD]

[DATA SECTION] is defined as *BEGIN [DATA] *END or
                         *BEGIN ([SECTION NAME]) [DATA] *END

[SECTION NAME] is defined as [WORD]

[DATA] is defined as [DATA TERM] or
                     [DATA TERM][DATA]

[DATA TERM] is defined as [WORD] or
                          [NUMBER] or
                          [SYMBOL]

[WORD] is defined as '[SYMBOL STRING]' or
                     [NON-SPECIAL SYMBOL STRING]
```

[NUMBER] is defined as [DIGIT STRING] or
[DIGIT STRING]. or
.[DIGIT STRING] or
[DIGIT STRING].[DIGIT STRING]

[SYMBOL STRING] is defined as [SYMBOL] or
[SYMBOL][SYMBOL STRING]

[NON-SPECIAL SYMBOL STRING] is defined as [NON-SPECIAL SYMBOL]
or [NON-SPECIAL SYMBOL][NON-SPECIAL SYMBOL STRING]

[DIGIT STRING] is defined as [DIGIT] or
[DIGIT][DIGIT STRING]

[SYMBOL] is defined as [SPECIAL SYMBOL] or
[NON-SPECIAL SYMBOL]

[SPECIAL SYMBOL] is defined as . or , or ' or (or) or * or
; or : or or # or - or
/ or = or > or < or & or | or ¬

[NON-SPECIAL SYMBOL] is defined as [LETTER] or
[DIGIT] or
[OTHER SYMBOL]

[LETTER] is defined as A or B or C or D or E or F or G or
H or I or J or K or L or M or N or
O or P or Q or R or S or T or U or
V or W or X or Y or Z

[DIGIT] is defined as 0 or 1 or 2 or 3 or 4 or 5 or 6 or
7 or 8 or 9

[OTHER SYMBOL] is defined as # or $ or % or ? or _ or @

APPENDIX B -- PLORTS TERMINALS

Communication with GENIRAS can be accomplished with statements typewritten at the user's terminal, with paper tapes fed into Teletype terminals, and with decks of punched cards brought to the computer system location. Since the primary means of communication with the system will be typewritten statements, a discussion of the use of punched cards and paper tapes has been deferred to the supplement to this manual dealing with Level 2 use of the system.

In addition to acquiring access to a terminal, a prospective user of GENIRAS must obtain both a valid problem specification number for the IBM System 360 and permission from the Service Group to use the PLORTS system. Once these details have been accomplished, the user's identification number will be recorded in the system for reference during signing on, as described below. The identification code may contain non-printing characters to give added security to stored records.

The following sections describe the use of the terminals for creating and changing PLORTS files of programs or data, for running these files, and for obtaining the results as output. As described earlier, there are several different categories of commands available to a user communicating with the system through a terminal. There are commands used in the file editing system for constructing and changing files, commands for formatting input statements, and commands in GENIRAL.

Logging In

The procedure referred to as logging in causes the terminal to be actively connected to the rest of the PLORTS system. Information stored in files in the system is not available to a user until he has properly logged in.

The procedure for logging in varies slightly depending on the type of terminal. These differences in procedure between Teletype Model 33 and IBM 2741 terminals have been noted wherever they arise.

1. Turn the terminal on. If the terminal has a direct connection, skip to step 4.

2. If the terminal has a separate data set (telephone coupler), press the TALK button, pick up the telephone, and dial the computer. Dial 333-4000 from a Teletype terminal. Dial 333-4001 from an IBM 2741.

3. The computer will respond with an answering tone. For the Teletype terminal, press the ORIG button on the data set, and then hang up. (Press the DATA button on an IBM 2741 terminal.)

4. Press the RETURN or RET key. If nothing happens, the system is completely unavailable or the terminal is not properly connected.

5. If the output is #XX TIME SHARING ON followed by the date and time, the PLORTS system is available.

6. If the output is #XX TIME SHARING OFF, the PLORTS system is temporarily unavailable. Try again later. (IBM terminals will type this as a string of meaningless characters, if PLORTS is unavailable.)

7. After #XX TIME SHARING ON and the date and time, the computer will type the message ENTER PS#,USER NAME. Respond by typing your identifying problem specification number and your name. The name and number should be separated by a comma, and should contain no blanks. (A user of the IBM 2741 might wish to precede this reply with the instruction #UC so that lower case letters will be accepted by the system.)

8. If the computer types BAD PS# OR NAME or PS# INVALID OR INACTIVE, it means that the PS# or name as entered, was not acceptable. Re-enter them both as

described in the previous step.

9. When the computer has accepted your name and PS# it will respond by typing out the number of PLORTS storage blocks that have been allocated and used under your PS#. After completing the above, the computer is ready to receive its first instruction.

10. When you have finished using the PLORTS system, you must indicate that you have finished by typing LOGOUT. The computer will respond by 'hanging up' the terminal.

NOTE: If you do not log out properly, your files will be available to the next person who signs on. That person will be able to open and use your files, and will be able to accrue charges for the use of computer time under your PS#. If the computer is not behaving properly when you are attempting to log in, it may mean that the previous user has forgotten to log out. Type LOGOUT, press the RETURN key, and try to log in again.

11. A response from the computer of #WAIT while you are trying to log in indicates that the PLORTS system is momentarily unavailable. (The #WAIT message is not typed by IBM 2741 terminals.)

Special Keys and Symbols

Both the Teletype Model 33 and the IBM 2741 terminals have special keys for communicating with the computer. Some of these special keys are used singly, while others are used in pairs to send commands to the computer. The use of these keys to perform similar functions differs greatly between the Teletype terminals and the IBM terminals. Because of this difference, the two types of terminals are separated in the following discussion.

The Teletype Model 33 or Model 35

The keyboard is a standard four-row teletypewriter keyboard that differs from a regular typewriter in the following respects.

1. Only upper case characters are available. No use of the SHIFT key is necessary. If the SHIFT key is pressed, one of the special characters will be obtained. These characters are marked as upper-case in the usual way.

2. The following characters differ from their markings on the Teletype keyboard:

SHIFT-K shown as a left bracket implies a ¢ (cent);
SHIFT-L shown as a \ implies a ¬ (not);
SHIFT-M shown as a right bracket implies _ (underscore);
SHIFT-N shown as ↑ or ^ implies | (or).

3. If the CTRL (control) key is pressed while other keys are being pressed, other special characters are obtained. Any keys and key combinations on the Teletype, except for those outlined below, should not be used without special authoriztion.

4. The standard control characters are obtained as follows:

RETURN (a non-printing character) causes the carriage to return to the beginning of the line. RETURN is the conventional method used to signal PLORTS that the line just typed is to be interpreted by the computer. Until RETURN is typed, the information typed on a line is not examined.

CONTROL-W or CTRL-W also causes the carriage to return to the beginning of the line. However, the information typed on the line is ignored by the computer.

SHIFT-0 backspaces the internal character reader by one space. It will not move the carriage back, however. Instead, an underscore or an arrow will print each time the key combination is struck. Since a new character typed for the same location in a line (as an incorrect character) will replace the incorrect character, the backspace is useful for correcting errors. For example, if the word TYPO is incorrectly typed as TYOP, SHIFT-0 should be typed twice followed by the letters PO. The terminal will type either TYOP__PO or TYOP←←PO. The word will be interpreted as TYPO.

5. Other control characters on the Teletype terminals are used to activate the multiplexor functions explained later in this section. They are listed here for reference. The key combination CTRL-Z cues the #NUM function. CTRL-I cues the #TAB and #STAB functions. CTRL-A cues the #COPY function, and RUBOUT cues the #COPYC function.

The IBM 2741

The IBM terminal resembles an electric typewriter with a standard keyboard. Both lower- and upper-case letters are available. The special keys and special functions of the IBM 2741 are as follows:

1. The following characters differ from their markings on the IBM 2741 keyboard:

SHIFT-1 shown as ± means a ¬ (not);
SHIFT-6 shown as a left bracket means < (less than);
SHIFT-! shown as a right bracket means > (greater than):
¦ shown as ! means | (or).

2. The standard control characters are obtained as follows:

RETURN causes the carriage to return to the beginning of the line. RETURN is the conventional method used to signal PLORTS that the line just typed is to be interpreted by the computer. Until RETURN is typed, the information on a line is not examined.

ATTN (printing / when it is struck) causes the computer to return to the beginning of the line in the same manner as the RETURN key. However, the information on the line is ignored by the computer.

BACKSPACE, like the backspace key on a typewriter, causes the carriage to move one space to the left each time it is struck. The BACKSPACE key can be used to backspace past typing errors to overstrike corrections. The newly typed letters will replace the letters in error. For example, if the word TYPO is incorrectly typed as TYOP, backspacing twice and typing the correction will give TYØØ. The computer will interpret the word as TYPO.

3. Only three multiplexor functions are available on the IBM 2741 terminals. First, #UC causes all alphabetic characters to be interpreted as upper-case. There is no control key combination associated with this function since it takes effect as soon as it is specified. The other two functions, #TAB and #STAB, are activated by the TAB key. These functions are explained later in this section.

Multiplexor Functions

This set of functions assists the user in formatting typed input lines. There are two instructions associated with most of the functions. The first gives the specification of the function; the second causes its use. During the specification, the multiplexor is told what function is being defined and in what way by means of an instruction which starts with the # character. Functions cannot be used until they are specified. The use of the functions is cued during programming, except as noted, by means of a control key or key compination.

#TAB N1 N2 N3 N4 (N2, N3, and N4 are optional and in ascending order) sets tab stops at columns N1, N2, N3, and N4. When the CTRL-I key combination is struck (the TAB key for IBM 2741 terminals), the carriage moves up to the next tabulator stop. When a line number has been created automatically by the #NUM function, the #TAB columns are counted with column 1 as the second position, after the line number. At all other times, the counting is done from the first printable column.

#STAB N1 N2 N3 N4 functions the same way as #TAB except that the spaces are inserted in the line only as it is seen by the computer and not in the line printed at the Terminal. A number from 1 to 4 is typed with the control key combination to indicate which tab stop has been selected. (#TAB and #STAB cannot be specified at the same time.)

#NUM N1 N2 provides automatic line numbering. N1 is the starting line number and N2 is the numbering increment. Available only on Teletype terminals, this function is activated by CTRL-Z.

#COPY provides a feature similar to the duplicate feature on a keypunch. After it has been specified, typing CTRL-A will cause the previous line to be copied from the column where CTRL-A is pressed to the next tab stop.

#COPYC allows single characters to be copied from the previous line. Each time the RUBOUT key is struck, one character is copied from the same location in the preceding line. When the RUBOUT and REPEAT keys are used together, strings of characters can be copied.

The following multiplexor functions do not have control key combinations associated with them. They become effective as soon as they are specified.

#UC causes all lower-case alphabetic characters to be translated to upper-case, Available only on IBM 2741 terminals, this function saves the trouble of holding the SHIFT key for long strings of upper-case characters.

#LONGLINE prints all the characters of a file line on one printed line.

#TAPEIN causes the paper tape reader to start.

#TAPEOUT causes any output for paper tape to be punched.

#TAPEOUTM, the same as #TAPEOUT, causes the paper tape to be punched in a format suitable for a manual reader.

#D nnn -- where nnn is one of the letter combinations T,S,N,C,CC,L,B, or U -- deletes the corresponding multiplexor function. T refers to #TAB or #STAB. S converts #STAB to #TAB. N refers to #NUM. C refers to #COPY. CC refers to #COPYC. L refers to #LONGLINE. B refers to the second bell. Finally, U refers to #UC.

File Editing

PLORTS files are referenced in the system by a name of eight or less characters, not including blanks, commas, or periods. Names with qualifiers also may be used. The qualifiers, each not more than eight letters long, are separated by periods. The qualifiers of a PLORTS name are the names of the subfiles contained within the PLORTS file.

A file name may be considered in terms of the implicit form: PS#.NAME.FILENAME. PS# represents the problem specification number, and NAME is the name with which the user signed on. FILENAME is the name of the PLORTS file referred to by the user, and may be qualified as stated. When John Q. Public signs on with 9999,PUBLIC, the names of all the PLORTS files created by Public are implicitly prefixed with 9999.PUBLIC. In this way, Public's files are available only to him.

Each line in a PLORTS file constitutes a record stored by its line number. Line numbers are of the form NNN.NNN, where N is a decimal digit. Fewer digits may be used in referring to a line, and the period may be omitted for integer values less than three digits long. The lines in a file are stored in increasing numerical order. The longest line that can be printed by an IBM terminal is about 126 characters long. The longest line that can be entered at the keyboard is 118 characters long. (Teletype lines are limited to about 80 characters.) Although lines longer than these limits cannot be completely displayed, they can be

filed and used by the batch processing system.

Two different sets of file editing commands are available, depending on the open or closed mode of the terminal. The terminal may be in only one of the two modes at any time.

Immediately after signing on, the terminal is in the closed mode, for which the set of commands is as follows:

CATALOG or CAT lists the names of files stored under PS#.NAME and the last date each file was opened. CATALOG may be typed by itself, or it may be followed by one or two file names. If one name follows CATALOG, then the names of the stored PLORTS files are listed alphabetically starting with that name. If two names follow CATALOG, only the file names between and including the two names are listed alphabetically.

RUN file1,file2,file3,.....causes the PLORTS files named in the command to be sent to the computer for execution. The files are sent in the order in which they are named. The list of names can extend over one line if each line to be continued ends with a comma after the last file name on that line. The lines in the files are passed on as 72-column character strings (to be handled like punched cards).

OPEN filename opens the file named. If the file named does not already exist, it is created.

O filename opens the file named only if it already exists. If O BASICS were typed as an instruction, and if a PLORTS file named BASICS did not exist, the command would be invalid. After OPEN or O is accepted, the terminal is in the open mode and the opened file can be modified by the addition, deletion, or replacement of lines.

LOGOUT disconnects the terminal from the PLORTS system. LOGOUT must be typed after each use of the system to prevent unauthorized access to files stored under the PS# and name used for logging in.

DEST FILE destroys the file named FILE.

TRACK lists the storage blocks allocated and the blocks used.

MSG prints any messages from the computer operator.

OPENMSG N (where N is a digit from 0 to 9), when followed by a LIST command, prints any messages and information about changes in the system.

TIME gives the time, the day of the week, and the date.

JOB JOBNAME (where JOBNAME is a complete eight-character job number) gives the status of the indicated job. Possible responses are NOT LOADED, IN SYSTEM, or OUT OF SYSTEM. This command may be used before logging in as a response to ENTER PS#,USER NAME.

The following commands apply to terminals in the open mode.

COPY filename copies the file named into the file that is open.

COPYS filename copies the file named into the open file, but copies only columns 1-72.

COPYE filename1.filename2. filename3...copies files from a different index level into the open file. If filename1 is a four-digit PS#, then filename2 might be a user's name, and filename3 might be a PLORTS file name.

DEL N1 N2 deletes the lines in a file numbered N1 through N2. If only one line number is given, then only the line with that number is deleted.

LIST N1 N2 lists the lines numbered N1 through N2 at the terminal. If only one line number is given, then all the lines in a file, starting with the given line, are listed. If no numbers are given, the entire contents of the open file are listed.

LISTN N1 N2 is the same as LIST, except that the line numbers are not listed.

LISTO N1 N2 is the same as LIST, except that the output is binary information for paper tape.

LAST lists the last line in the open file.

ENTER causes everything entered at the console to be appended to the end of the open file. The addition of lines will continue until the command EXITTT is typed. Only multiplexor functions and the EXITTT command are recognized by the computer after ENTER is given (no line numbers can be used).

EXITTT ends the effect of the ENTER command.

CLOSE closes the open file and saves it. This command returns the terminal to the closed mode.

Individual lines may be entered in an open PLORTS file by typing a line number followed by a single space and the desired information. The line number may take any of the permissible forms described earlier. Valid line numbers might be 32, 43.016, .3, 4., or 100. The line numbers 1000 and .0345 would be invalid. When the line is entered, the line number is expanded automatically to the six-digit form, and the line is filed in its proper numerical position. If a line with the same number is already present, it is replaced by the new line. Spaces typed before the line number are ignored when the line is entered. However, spaces after the line number are part of the line. The line number must always be followed by at least one blank.